Traditional Machine Learning Methods on Graphs

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

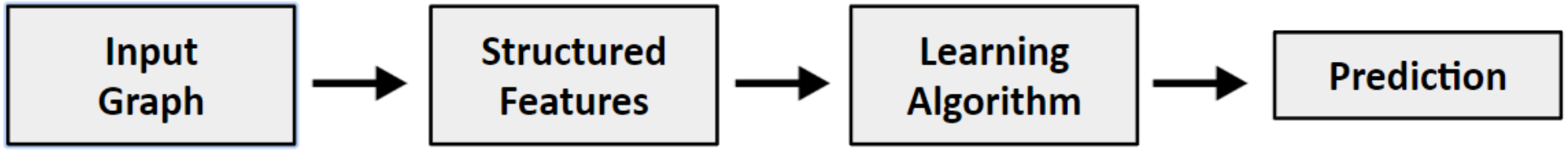
네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

- Node embeddings introduction
- Node embedding methods
  - Random-walk based methods
    - Deepwalk
    - Node2vec
  - Proximity-based methods
    - LINE
- Applications and Limitations

- We can extract node, edge, graph-level features from the graph, then learn a model to map the features to the desired labels.



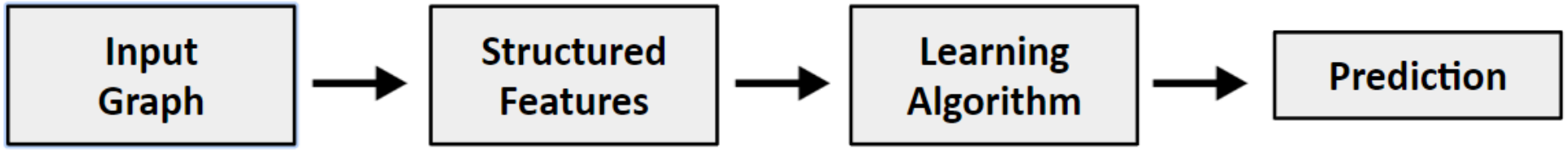
Feature Engineering

- Node feature
- Edge feature
- Graph feature

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

- Graph Representation Learning aims to generate graph representation vectors that describe graph's structure. So we don't need to do feature engineering every single time.



Feature Engineering

Representation Learning

learn the features by itself

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

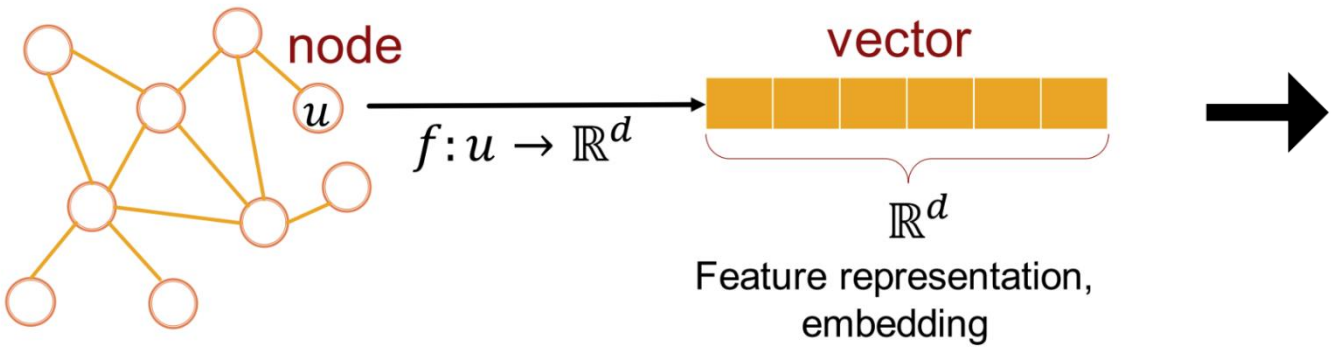
Why Embeddings?

- Graph Representation Learning:

  Learn efficient task-independent feature for machine learning with graphs.

  → Map nodes into an embedding space

- Similarity of embeddings between nodes indicates their similarity in the network. (Similar to word embedding)
- For simplicity, no node features or extra information is used



Tasks

- Node classification/regression
- Edge prediction
- Graph cls/reg
- Clusternig
- …

➢ Goal:

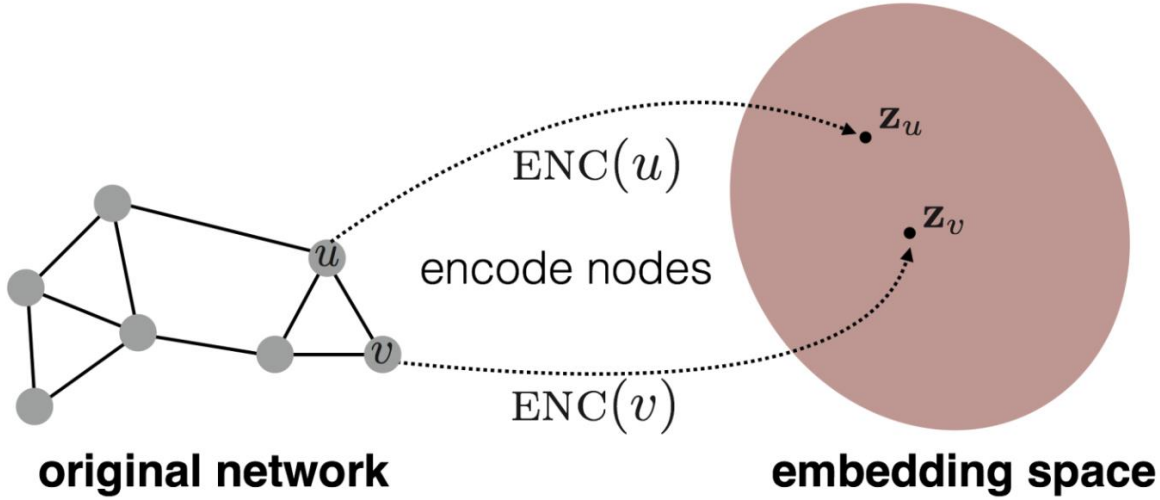
Encode nodes → similarity in embedding space (dot product) ≈ similarity in the original graph

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

We need to define:

ENC(u)

Similarity(u, v)



Learning Node Embeddings

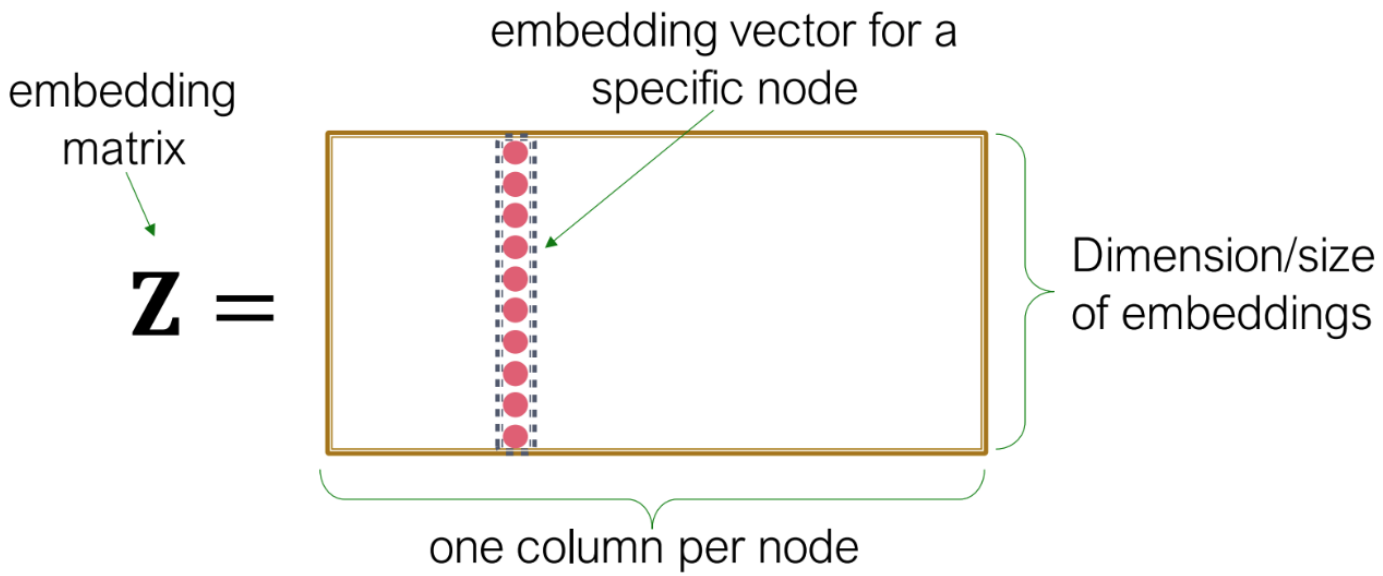
- Encoder: node → embedding (d-dimensional vector, ENC(v) = Z_v)
- Decoder: embedding → similarity score (dot product)
- Define a node similarity function:

  whether 2 nodes are close in the graph, and how close are they?

- Optimize → $\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$

- Simplest encoding approach: Build a embedding lookup table

$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$



$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$$

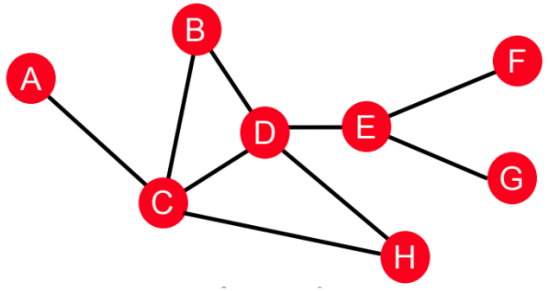
matrix, each column is a node embedding (this is what we want to learn)

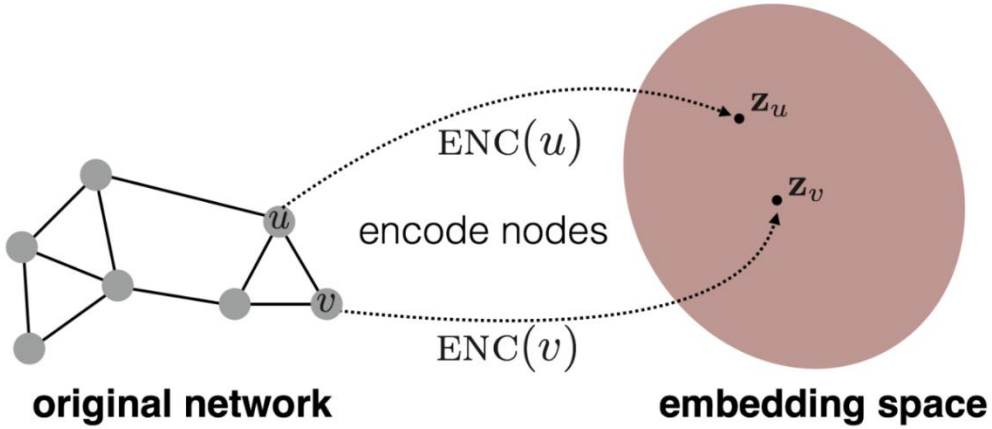
$$v \in \mathbb{I}^{|\mathcal{V}|}$$

indicator vector, all zeros except a one in column for node v

- Each node is assigned a unique embedding vector → we **directly optimize the embedding of each node**
- Embedding is optimized to maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for each similar node pairs (u, v)
- Key choice: How to define node similarity? Should 2 nodes have similar embedding if:
  - They are linked?
  - They share neighbors?
  - They have similar structure roles?

→ **Random Walks**

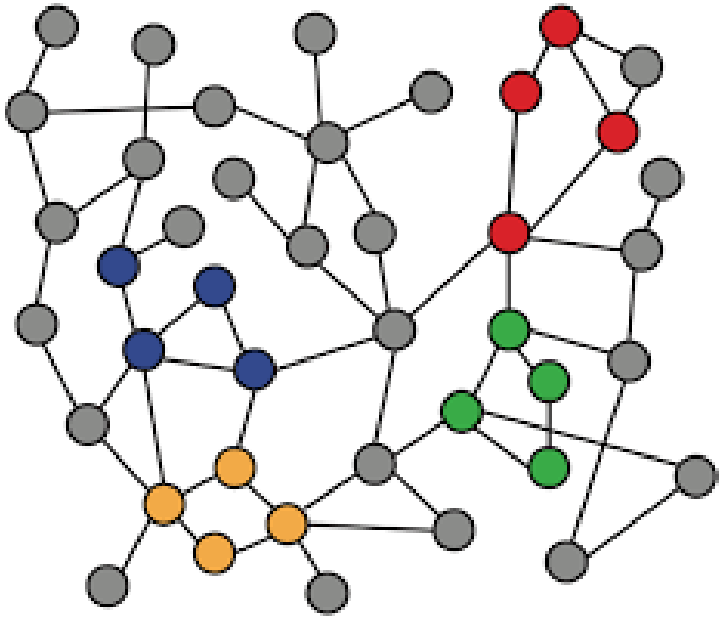




Note on Node Embeddings

- This is unsupervised/self-supervised way of learning node embeddings.
  - Not using node labels
  - Not using node features
  - Directly learn a set of coordinates (the embedding) of a node, so that some feature of the network is preserved.
- These embeddings are task independent
  - They can be used in different downstream predictions.

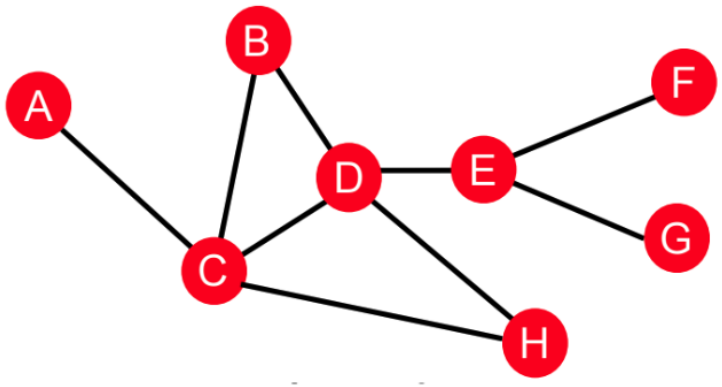
- Random Walk
- Learning Objective & Method
- node2vec

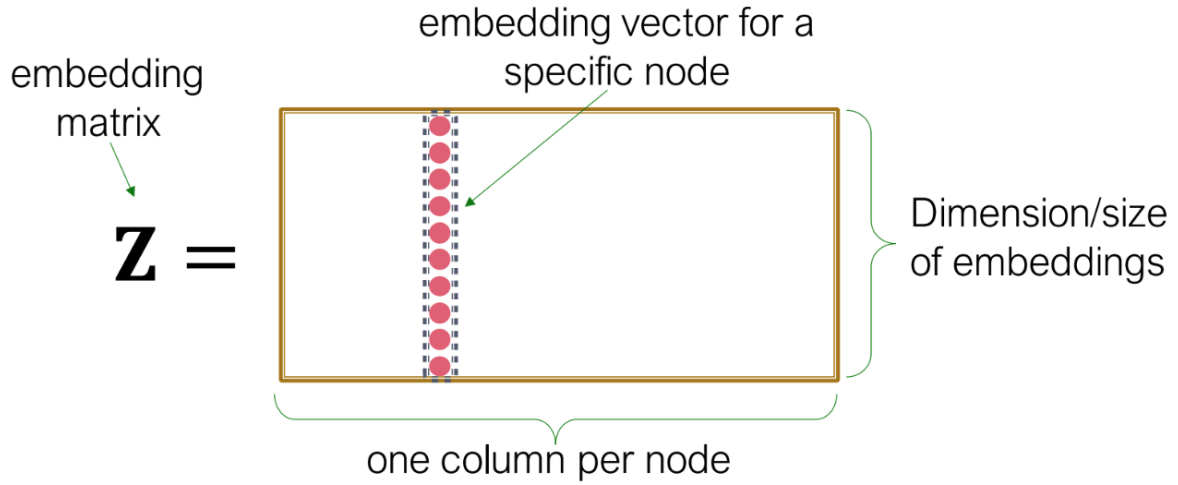


Jure Leskovec – Stanford CS224W: Machine Learning with Graphs

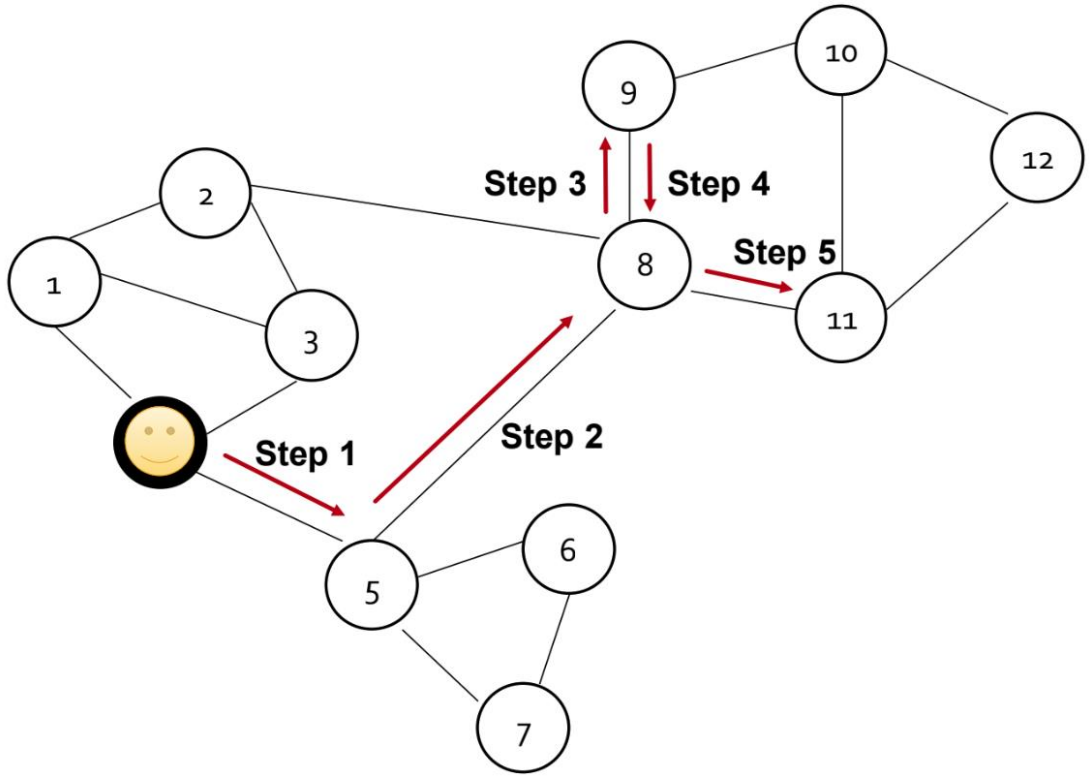
- We want to learn the embedding for every node $\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$ such that:

$$\underset{\text{in the original network}}{\text{similarity}(u, v)} \approx \underset{\text{Similarity of the embedding}}{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}$$

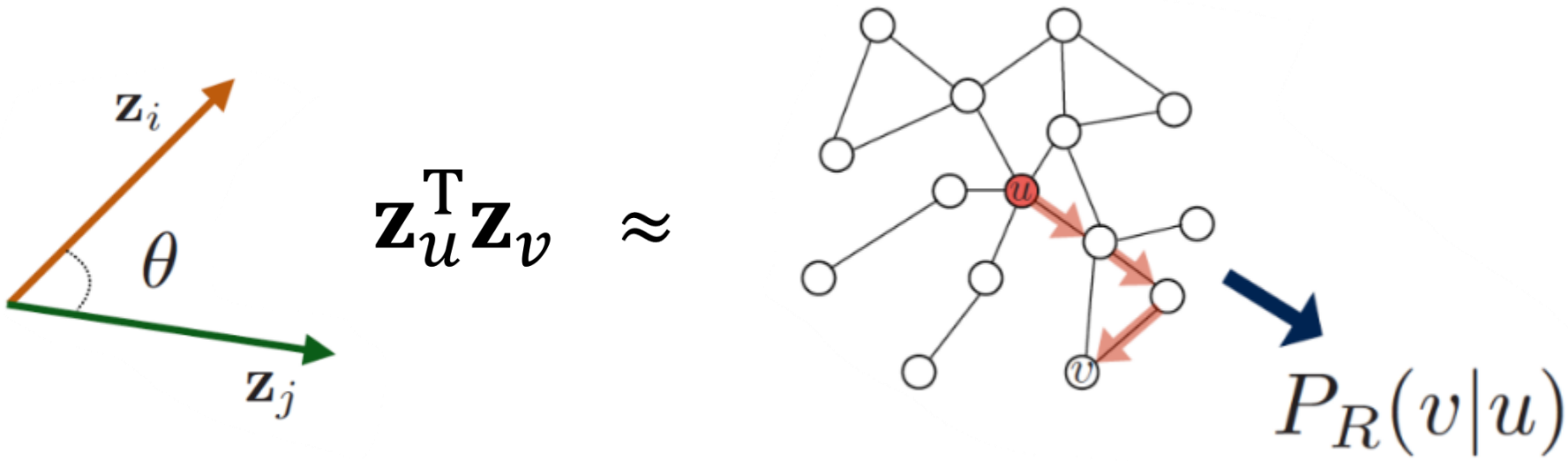




- Given a graph and a starting point, we select a neighbour of it at random, and move to this neighbour.
- Then, we select a neighbor of this point at random, and move to it,…
- The random sequence of points visited this way is **a random walk on the graph**



- Estimate probability of visiting node v on a random walk starting from node u with strategy R : $P(v \mid \mathbf{z}_u)$
- Optimize node embeddings to learn the statistics from random walks



Why using Random Walks?

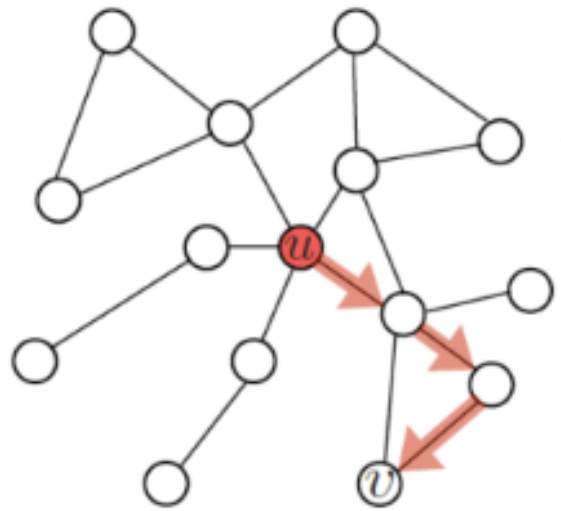
- We can simply define nodes are similar if there are connected, why random walks?

- **Expressivity**:

  Random walk incorporates both local and higher-order multi-hop nehgiborhood information

- **Efficiency**:

  Don’t need to consider all node pairs at training state; only need to consider node pairs on the random walks



Feature Learning as Optimization

- Log-likelihood objective: $\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$

  $N_R(u)$ is the neighborhood of node u by random walk strategy R

- Equivalently, our loss function:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}$$

- Run short fixed-length random walks starting from each node u
- For each node u, collect the visited node set $N_R(u)$
- Find embeddings $\mathbf{z}_u$ that minimizes L

Negative Sampling in Optimzation

- Problem: Expensive in summing over nodes ($O(|V|^2)$)

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

- **Solution:** Negative Sampling (Softmax → Sigmoid)

$$\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right) \approx \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)\right) - \sum_{i=1}^{k} \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_{n_i})\right), n_i \sim P_V$$

random distribution over nodes

- Higher k gives more robust estimates.
- In practice k = [5, 20]

- Goal: Optimize $\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
- (Stochastic) Gradient Descent: a simple way to minimize L

- **SGD Algorithm:**
  - Initialize z_u at some randomized value for all nodes u.
  - Iterative until L converges:
    - Sample a node u, for all v calculate the derivative $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$
    - For all v, update:

$$z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$$

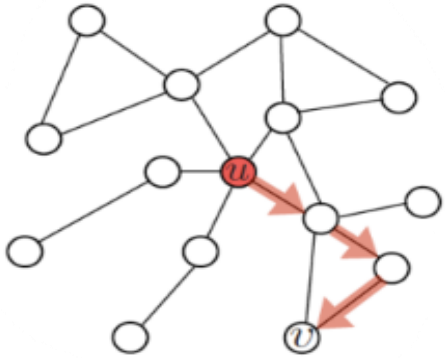
How should we randomly walk?

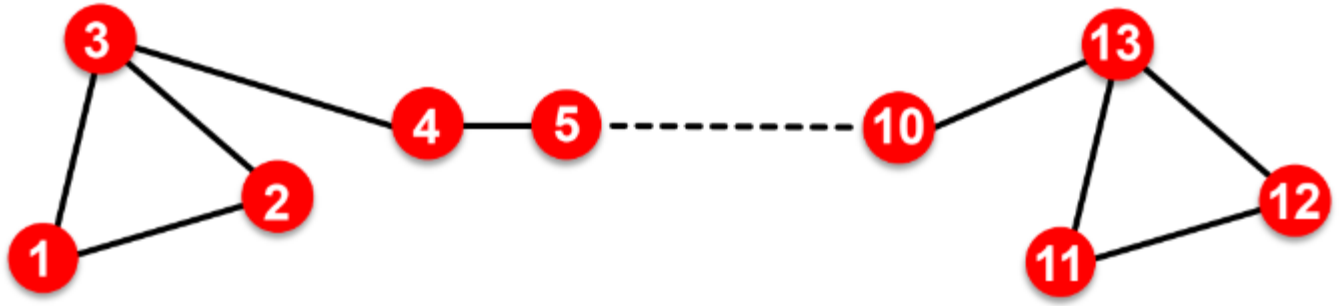
- What strategies should we use to run the random walks?
- Simplest idea:

  Run fixed-length, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al., 2013)

- **Issue:**

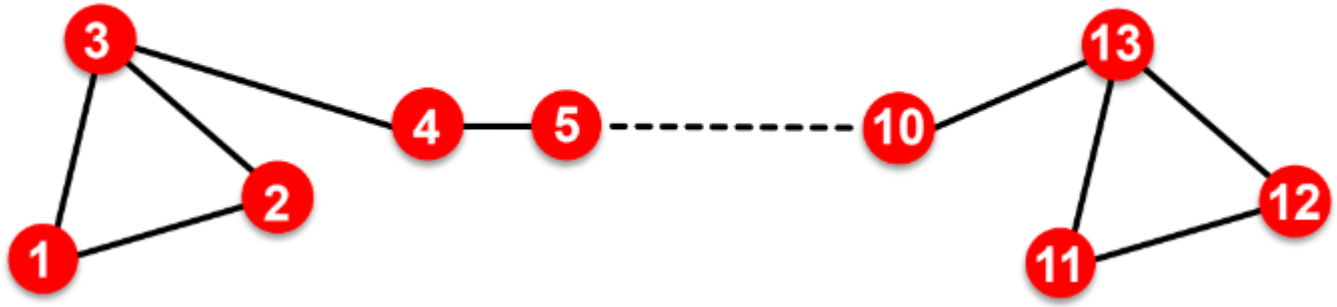
  Not rich enough to embed nodes from same network community as well as nodes with similar structural roles





node2vec: Biased Walks

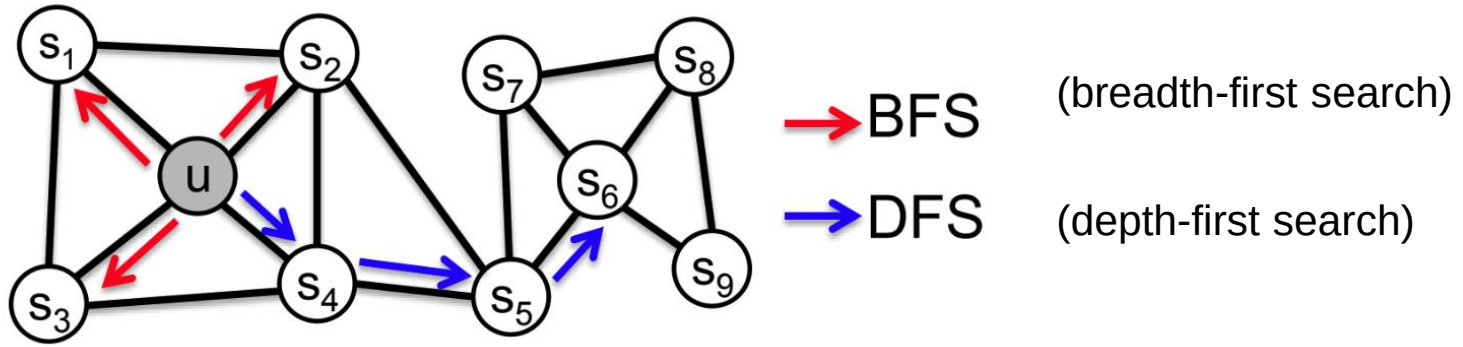
- **Goal**: Embed nodes with similar network neighborhoods close in the embedding space.
- **Key**: Develop biased 2nd order random walk strategy R to generate richer node embeddings.
- **Learning method**: Maximum likelihood optimization problem



Hope to embed well on community networks and similar structural roles

node2vec: Biased Walks

- **Idea**: Use flexible, biased random walks that can trade off between local and global views of the network (Grover and Leskovec, 2016).
- Two classic strategies to define a neighborhood $N_R(u)$ of a given node u



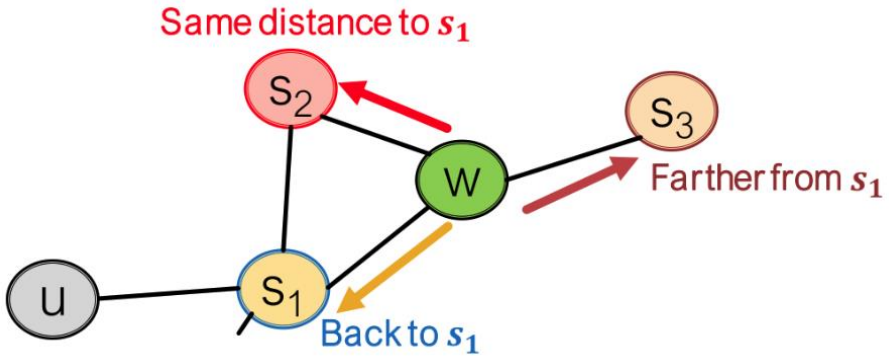
- Walk of length ($N_R(u)$ of size 3):

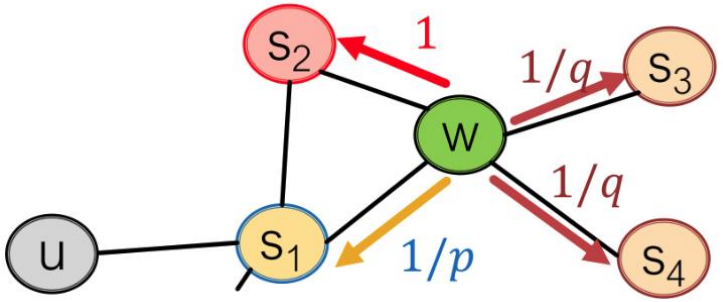
$$N_{BFS}(u) = \{s_1, s_2, s_3\}$$ **Local view**

$$N_{DFS}(u) = \{s_4, s_5, s_6\}$$ **Global view**

Biased Random Walk

- Biased 2nd-order random walks explore network neighborhood:
- Walker just traversed edge (s1, w) and is at w, now he can go:





1/p, 1, 1/q are unnormalized probs.
p: return parameter
q: "walk away" parameter

- BFS-like walk: Low value of p
- DFS-like walk: Low value of q

$N_R(u)$ **are the nodes visited by the biased walk.**

node2vec algorithm

- Compute random walk probabilities
- Simulate r random walks of length l starting from each node u
- Optimize the node2vec objective using stochastic gradient descent

→ The training process is same as DeepWalk, except the walking strategy

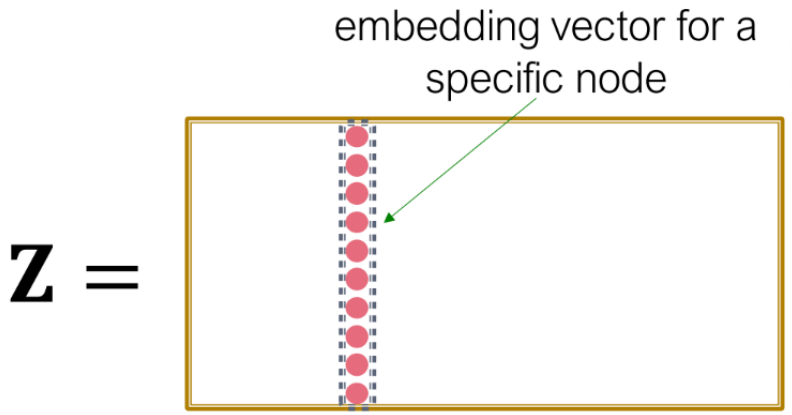
Summary of Random Walks

- **What can we do now?** → We can get good node embeddings that distances in embedding space reflect node similarities in the original graph network.
- **Different method for node similarity:**
  - Naive: similar if 2 nodes are connected
  - Local, global neighborhood overlap
  - Random walk approaches
- **Which method should we use?**
  - No one method wins in all cases
  - choose node similarity that matches your application

Random Walks for Shallow Encoding:

➢ **Goal**: Generate a lookup table for node embeddings

1. Run random walks for each node
2. Collect the set of visited nodes for each node on the walks
3. Optimize the embeddings z_u using stochastic gradient descent



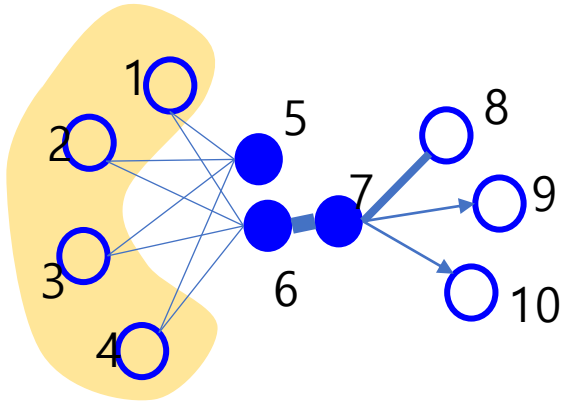
Proximity Preservation: LINE

- Has a clear objective function
  - Preserve the first-order and second-order proximity between the vertices
- Very scalable
  - Effective and efficient optimization algorithm through asynchronous stochastic gradient descent
  - Only take a couple of hours to embed network with millions of nodes, billions of edges on a single machine

Tang et al. LINE: Large-scale Information Network Embedding. WWW'15

First-order Proximity

- The local pairwise proximity between the vertices
  - Determined by the observed links
- However, many links between the vertices are missing
  - Not sufficient for preserving the entire network structure

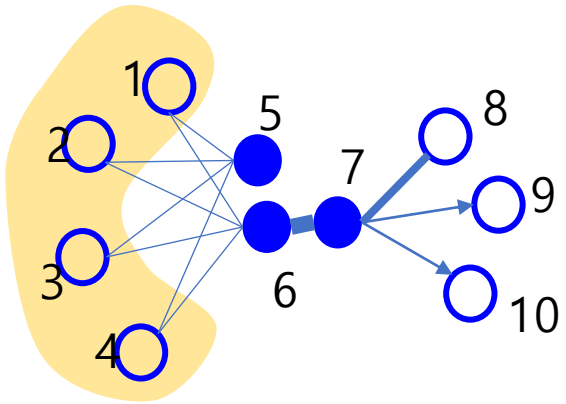


Vertex **6** and **7** have a large first-order proximity

Second-order Proximity

- The proximity between the neighbourhood structures of the vertices
- Mathematically, the second-order proximity between each pair of vertices (u,v) is determined by:

$$\hat{p}_u = (w_{u1}, w_{u2}, \dots, w_{u|V|})$$
$$\hat{p}_v = (w_{v1}, w_{v2}, \dots, w_{v|V|})$$



Vertex **5** and **6** have a large second-order proximity

$\hat{p}_5 = (1,1, 1,1,0,0,0,0,0,0)$

$\hat{p}_6 = (1,1, 1,1,0,0,1,0,0,0)$

Preserving the First-order Proximity

- Given an ***undirected*** edge (v_i, v_j), the joint probability of v_i, v_j

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

$\vec{u}_i$: Embedding of vertex v_i

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(i',j')} w_{i'j'}}$$

- Objective:

$$O_1 = d(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot))$$

KL-divergence

$$\propto -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

Preserving the Second-order Proximity

- Given a directed edge (v_i, v_j), the conditional probability of v_j given v_i is:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}$$

$\vec{u}_i$: Embedding of vertex *i* when *i* is a source node;
$\vec{u}_i'$: Embedding of vertex *i* when *i* is a target node.

- Objective:

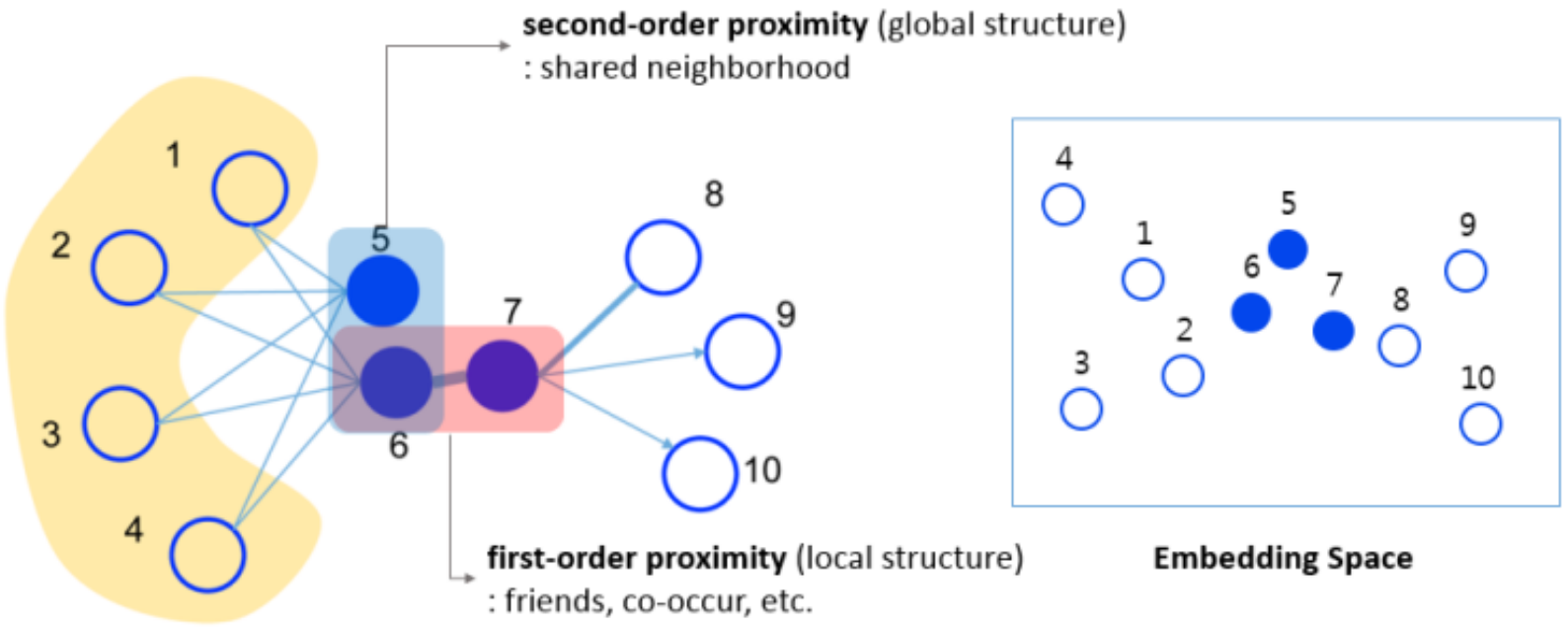
$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i))$$

$$\propto -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i)$$

λ_i: Prestige of vertex in the network $\lambda_i = \sum_j w_{ij}$

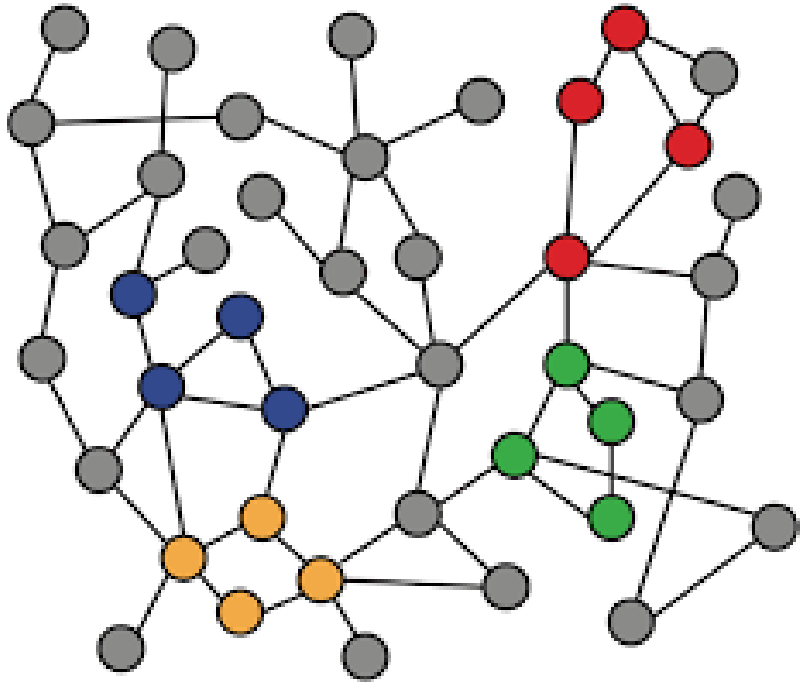
Preserving both Proximity

Preserve both local & global network structure



3. Applications and Limitations of Shallow Encoding

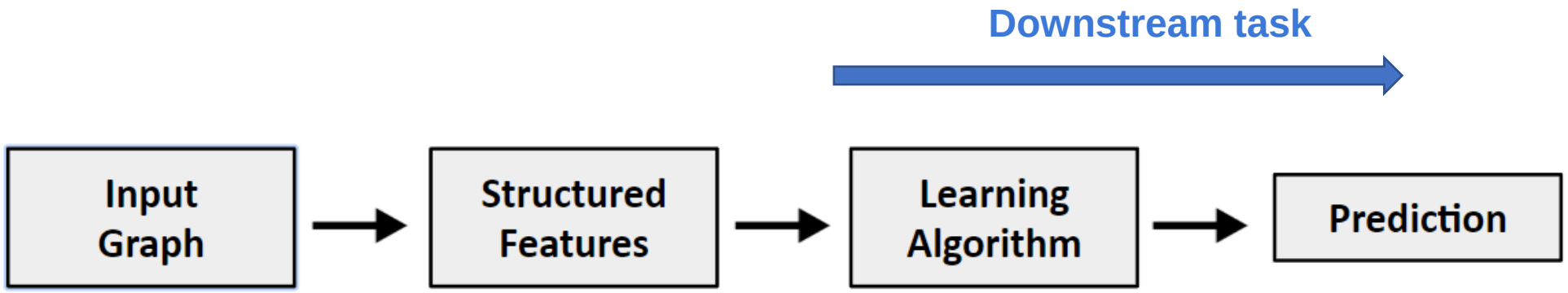
- Application of Node Embeddings
- Limitations of Shallow Encoding
- Transductive, Inductive Learning



http://web.stanford.edu/class/cs224w/

What After Node Embeddings?

- Once we have node embeddings (independent to task), we can continue to the downstream prediction.



Shallow Encoding

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

How to Use these Node Embeddings?

➢ **Given a node v_i in a graph, we have it's embedding Z_i, we can do:**

(1) Clustering/community detection: Cluster $\boldsymbol{Z_i}$

(2) Node classification: Predict label of node i based on $\boldsymbol{Z_i}$

(3) Link prediction: Predict edge (i, j) based on ($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$)

Concatenate: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g([$\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$])

Hadamard: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g($\boldsymbol{Z_i} * \boldsymbol{Z_j}$) (per position product)

Sum/Average: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g($\boldsymbol{Z_i} + \boldsymbol{Z_j}$)

Distance: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g(|$\boldsymbol{Z_i} - \boldsymbol{Z_j}$|2)

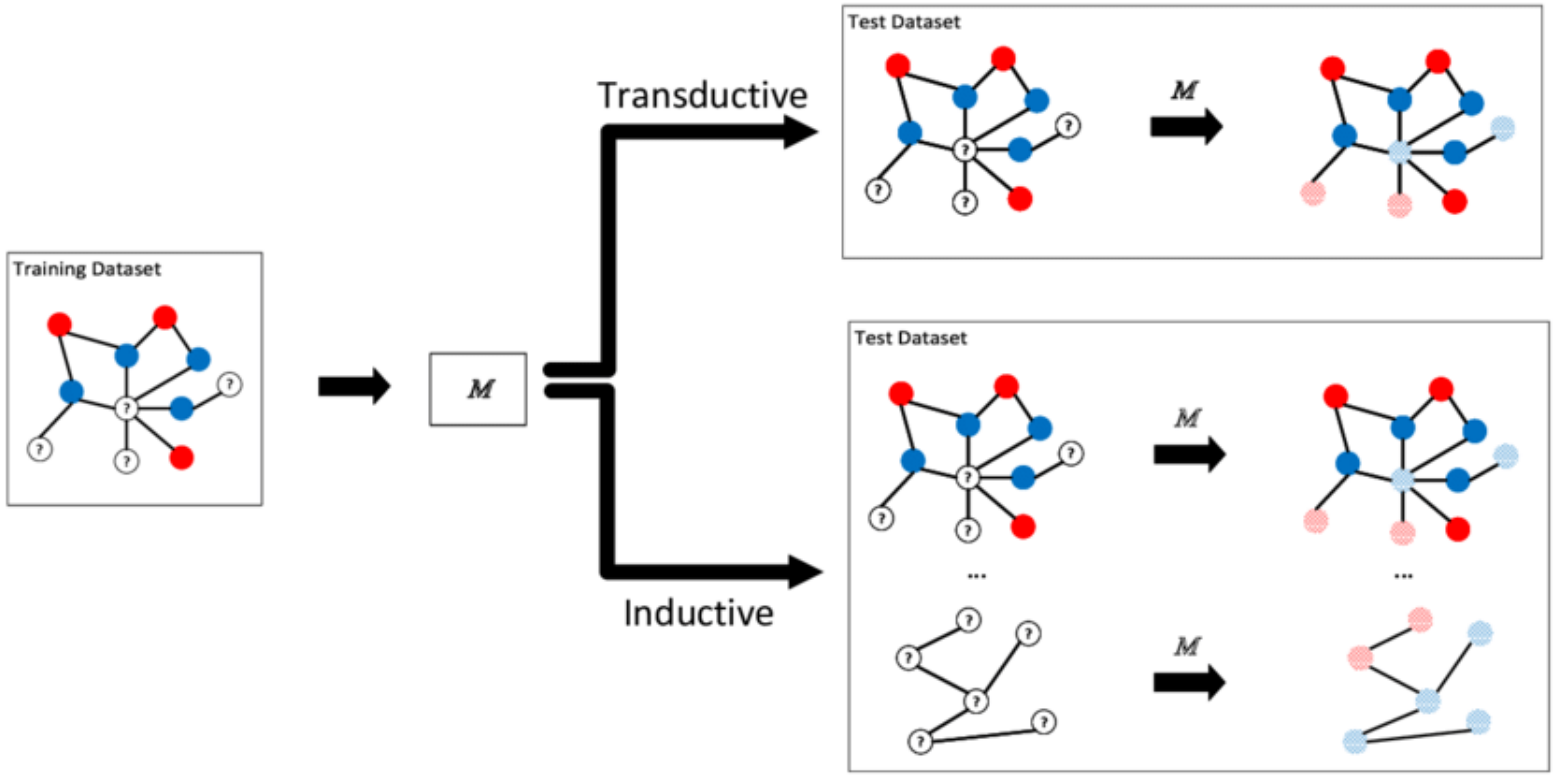
(4) Graph classification: aggregate node embeddings to form graph embedding $\boldsymbol{Z_G}$. Predict graph label based on graph embedding $\boldsymbol{Z_G}$.

Limitations of Shallow Encoders

- Limitations of shallow embedding methods:
- O(|V|) parameters are needed:
  - Every node has its own unique embedding
  - No shared parameters between nodes
- Inherently “transductive”:
  - Cannot generate embedding for nodes that are not seen during training
- Do not incorporate node features:
  - Nodes in many graphs have node features that we can and should leverage

Transductive vs. Inductive Learning

- In transductive learning, for new coming graphs, we have to train from scratch to get the embeddings. However in inductive learning, the way we create embeddings can be generalized to unseen graphs.
- Inductive learning on graphs → Deep Encoder → Graph Neural Networks (GNNs)



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA